(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 860 776 A1

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
26.08.1998 Bulletin 1998/35

(51) Int. Cl.[6]: G06F 9/46

(21) Application number: 98102297.3

(22) Date of filing: 10.02.1998

(84) Designated Contracting States:
AT BE CH DE DK ES FI FR GB GR IE IT LI LU MC NL PT SE
Designated Extension States:
AL LT LV MK RO SI

(30) Priority: 19.02.1997 JP 51090/97

(71) Applicant: Hitachi, Ltd.
Chiyoda-ku, Tokyo 101-0062 (JP)

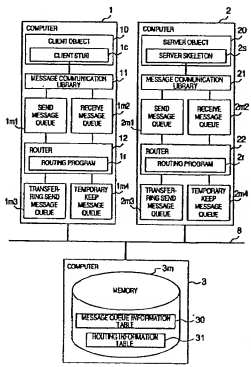
(72) Inventors:
- Aoshima, Tatsundo
Sagamihara-shi (JP)
- Hashimoto, Tetsuya
Suginami-ku, Tokyo (JP)

(74) Representative:
Strehl Schübel-Hopf & Partner
Maximilianstrasse 54
80538 München (DE)

(54) Inter-object communication method

(57) A method for converting an interface definition description and an inter-object communication method, in which a service provider can control a message delivery method to realize communication based not on object designation but on service designation. In the methods, a client stub (1c), a server skeleton (2s) and a routing program (1r,2r) are generated from an interface definition defined by the service provider. A client object (10) calls the client stub, and sends a message whose send destination is specified by an interface name. The server skeleton registers a provided service in a routing information table (31). A router (12,22) determines a server object (20) as the send destination on the basis of a routing program associated with the interface name of the send message, a message queue information table (30) and the routing information table (31), and sends the message.

FIG.1



EP 0 860 776 A1

Description

BACKGROUND OF THE INVENTION

The present invention relates to an inter-object communication method in an application which uses a plurality of objects to be operated on a single computer or on a plurality of computers connected in a network and more particularly, and also relates to a method for converting an interface definition in such an application.

Conventional inter-object communication method in an application which uses a plurality of objects to be operated on a single computer or on a plurality of computers connected in a network, includes a common object request broker architecture (CORBA)(Architecture and Specification Revision 2.0, July 1995, OMG) In the Object Management Group (OMG). In the CORBA, when interface definition language (IDL) is used, an object interface can be defined independently of a programming language for describing the objects.

When an interface definition described in the IDL is converted by an IDL compiler, there are automatically generated a client stub (term often used in the above CORBA) associated with a client program to play a role in sending a message from the client program to a server program as well as a server skeleton (term often used in the above CORBA, which is also a template for processing description) corresponding to a description part of the server program other than an original processing description part thereof, thus enabling relief of programmer's burden.

Actual communication between the programs is carried out on the basis of the object request broker (ORB) technique.

The client program searches for a communication destination object on the basis of the name, acquires as its searched result an object reference (which is information indicative of the presence location of the object (server program)) associated with the server program corresponding to the object in a one-to-one relationship; whereas, the client program transmits a message to the server program by calling a client stub generated from the interface associated with the object reference.

Thereby a programmer of the client program can describe the original-processing description part of the application program without recognizing the physical position of the server program or the actual protocol.

Meanwhile, a programmer of the server program is only required to describe only the original processing of the application program without recognizing the actual communication protocol.

SUMMARY OF THE INVENTION

It is therefore an object of the present invention to provide a programming method centered on the original processing (service) provided by a server object while securing the above advantages of a CORBA, and also to provide an inter-object communication method.

The object includes five sub-objects which follow.

First one of the sub-objects is to enable communication with the server object which provides a service without any need for specifying the object itself, by specifying the service.

Second one of the sub-objects is to suitably deliver a communication message to the server object which provides the service.

Third one of the sub-objects is to eliminate the need for modifying the client object at the time of adding, deleting or modifying the server object which provides the service.

Fourth one of the sub-objects is to enable the service provider and the programmer of the server object to control the message distribution method.

Fifth one of the sub-objects is to centralizedly manage a program module associated with one application to simplify its installing operation.

In accordance with an aspect of the present invention, the above objects can be attained by providing a method for converting an interface definition description in an inter-object communication system wherein a client object operating on either one of a plurality of computers in a network or on a single computer sends a message to a server object operating on the same or another computer to execute a predetermined processing operation.

In this invention, the method includes the step of causing an interface definition converting program to convert the interface definition description described in an interface definition language to thereby generate a client stub, a server skeleton and a routing program. The interface definition description includes one or more of data necessary for the predetermined processing operation, a method definition description of a processing result, and a message description for specifying a format of the message to be sent in response to the method definition description. The client stub is called to cause the client object to send the message. The server skeleton includes a server registration method for storing, in routing information memory means, routing information for specifying the format of the message which the server object itself can process when started and also includes a method for describing the predetermined processing operation to be executed when the server object receives the message. The routing program judges whether or not the processing of the server object associated with the routing information of the message is possible through comparison between the message and the routing information.

In accordance with another aspect of the present invention, there is provided an inter-object communication system wherein a client object operating on either one of a plurality of computers in a network or on a single computer sends a message to a server object operating on the same or another computer to execute a predetermined processing operation. In this method, a

client stub, a server skeleton and a routing program generated through the above conversion are stored in the computers. When the server object is started, a server registration method is called and routing information is stored in routing information memory means to cause the client object to call the client stub to send the message and to hold the sent message in a send message queue. A router extracts the message from the send message queue, to determine a send destination object according to the routing program and to add the received message in a receive message queue associated with the send destination object. The server skeleton extracts the message from the receive message queue and the server object executes the predetermined processing operation of the message extracted by the server skeleton.

In the method, the routing program adds a queueing option description including a message maximum number designation and a queueing time designation to a message description of the interface definition description, compares the message and the routing information to judge whether or not the processing operation by the server object associated with the routing information of the message is possible and, in the case of presence of the message maximum number designation, performs the queueing option judging operation.

In the inter-object communication system, the router keeps the message extracted from the send message queue in a temporary keep message queue, and when another message has the same contents as the kept message, the router integrates the messages into a single message, and when the number of messages within the temporary keep message queue exceeds the message maximum number or when the queueing time elapses after the message is queued, the router adds the message to a receive message queue associated with the send destination object.

In the method, in place of the routing program, such a custom routing program is generated as to compare the message and the routing information to judge whether or not the processing operation by the server object associated with the routing information of the message is possible and to perform the matching option judging operation.

In the inter-object communication system, further, the client stub, the server skeleton and the routing program are combined into a single server object to store the service object in a service object memory means; the server object is extracted from the service memory means to call the client stub before the client object sends a message; the server object is extracted from the service memory means to call the routing program before the router determines the send destination object; and the server object is extracted from the service memory means to call the server skeleton at the time of starting the server object.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 shows a general arrangement of an embodiment of the present invention;

Fig. 2 shows a structure of an IDL compiler and a relationship thereof with an interface definition file, a client stub, a server skeleton, a routing program;

Fig. 3 is a flowchart for explaining how to generate the client stub;

Fig. 4 is a flowchart for explaining how to generate the server skeleton;

Fig. 5 is a flowchart for explaining how to generate the routing program;

Fig. 6 shows an exemplary structure of the interface definition file;

Fig. 7 shows an exemplary structure of the client stub;

Fig. 8 shows an exemplary structure of the server skeleton;

Fig. 9 shows an exemplary structure of the routing program;

Fig. 10 shows a structure of a message queue;

Fig. 11 is a structure of a message queue information table;

Fig. 12 shows a structure of a routing information table;

Fig. 13 is a flowchart for explaining a client stub process;

Fig. 14 is a flowchart for explaining a server skeleton process;

Fig. 15 is a flowchart for explaining a routing process;

Fig. 16 shows an arrangement of an inter-object communication system using a dynamic service object loader; and

Fig. 17 shows a structure of a server object.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

An embodiment of the present invention will be detailed below with reference to the drawings.

Referring first to Fig. 1, there is shown an entire arrangement of an embodiment of the present invention.

In the present embodiment, computers 1, 2 and 3 are connected in a network 8.

A client object 10 as a program for requiring a service to a server is present in the computer 1, and, a server object 20 as a program for executing a service in response to a request from a client is present in the computer 2 independently from each other.

In the inter-object communication of the present invention, however, it is possible that a client object 10 and the server object 20 are both present in the same computer 6 as shown in Fig. 16, and it is also possible that a plurality of client objects 10 and a plurality of server objects 20 are mixedly present in a single com-

puter.

In addition, it is also possible to place a memory 3m of the computer 3 in the computer 1 or 2.

Fig. 2 schematically shows how to generate a client stub 1c for sending a message from the client program 1o to the server program, a server skeleton 2s for serving as a template for processing description of the server program, and a routing program 1r for judging whether or not the service can be processed by the server object on the basis of the received message, from an interface definition file 4 prepared in an interface definition language (IDL) by a service provider with use of an interface definition conversion program 5 (IDL compiler 5).

The IDL compiler 5 includes a lexical analyzer 50, a parser 51, a client stub generator 52, a server skeleton generator 53 and a routing program generator 54.

The lexical analyzer 50 analyzes a token while reading the interface definition file 4 and provides information on the read token to the parser 51.

The parser 51 performs its analyzing or parsing operation using the above information and calls the client stub generator 52, server skeleton generator 53 and routing program generator 54.

Shown in Fig. 6 is a structure of an example of the interface definition file.

The interface definition file 4 is a further extension of the IDL description used in the CORBA.

The interface definition file 4 has a native method description 40 for calling an external method, and method definitions 41 and 42 corresponding to services provided by the server object.

The method definition 41 has a method declaration (corresponding to a description "void search... Exception" given under Method Definition 1) used in the CORBA, a message description 410 and a matching option description 411 for performing a strict matching operation based on user's instruction as an option.

The matching option description 411 is described in a language independent format as in the case where the CORBA IDL description is language-independent.

The example of Fig. 6 shows the message description 410 indicating that a communication message has a pattern of "search $keyword", where a token starting with $ is a parameter.

The matching option description 411 is one which judges whether or not the parameter "$keyword" can be processed with use of the method "isSupportCategory()" declared in the native method description 40.

The message description 410 may include a queueing option description 4100 as an option.

In the example of Fig. 6, the maximum number 'maximum' of messages handleable in the message integrating operation is specified to be 100, and the longest message queueing time is specified to be 60 seconds.

The message integration will be described later.

Fig. 7 shows a structure of an example of the client stub compiled and generated by the IDL compiler from the interface definition file 4 shown in Fig. 6 as an example. The generated client stub 1c is described in source level language. In the example of Fig. 7, C++ is used as a programming language after the compilation.

The client stub 1c in compiled code is prepared by compiling a client stub header file 1cf1 and a client stub source file 1cf2.

The client stub header file 1cf1 includes messaging method declarations 1cf11 and 1cf12 called by the client object 10.

The client stub source file 1cf2 includes messaging method definitions 1cf21 and 1cf22.

Shown in Fig. 3 is a flowchart for explaining a client stub generating process by the client stub generator 52.

A client stub generating process 520, when called by the parser 51, analyzes the interface definition file 4 sequentially from the method definition 41 therein, repeats its subsequent processing operations on the basis of a judgement step 521 on whether or not there is a method definition, and if there is no definition, terminates its operation at an end step 527.

In compiling the method definition 41, at a messaging method declaration code generation step 522, there are generated the messaging method declaration 1cf11 and a protocol type declaration part (which corresponds to "void DigitalLibrary... Exception" given below the Messaging Method Definition. The declaration part in the C++ language is also called a prototype declaration part) of the messaging method definition 1cf21.

Next, at a message pattern string definition code generation step 523, codes for preparation of a message string not containing a parameter are generated in the messaging method definition 1cf21.

When a parameter is present in the message string, the system repeats the operation of a parameter setting code generation step 525 on the basis of a judgement result of the step 524 indicative of the presence of absence of a parameter. At the parameter setting code generation step 525, the system generates in the messaging method definition 1cf21 codes for addition of a parameter string to the above message string.

Finally, at a message sending code generation step 526, the system generates in the messaging method definition 1cf21 codes for calling of a message sending function of a message communication library 11 (see Fig. 11) with use of an interface name ("DigitalLibrary") and the message string as parameters.

Fig. 8 shows a structure of an example of the server skeleton compiled and generated by the IDL compiler from the interface definition file 4 of Fig. 6. The generated server skeleton 2s is described in source level language. Even in the example of Fig. 8, C++ is used as a programming language after the compilation.

The server skeleton 2s in compiled code is generated by compiling a server skeleton header file 2sf1 and a server skeleton stub source file 2sf2.

The server skeleton header file 2sf1 includes a registration method declaration 2sf10 to be called for regis-

tration of the server object 20 in the routing information table (see Fig. 1) at the time of activating the server object 20 and messaging handler declarations 2sf11 and 2sf12.

The server skeleton stub source file 2sf2 includes a registration method definition 2sf20 and messaging handler definitions 2sf21 and 2sf22.

Fig. 4 is a flowchart for explaining the operation of a server skeleton generation process 530 by the server skeleton generator 53.

The server skeleton generation process 530 is called by the parser 51.

At a registration method declaration code generation step 531, the system generates the registration method declaration 2sf10 and a prototype declaration part (corresponding to "void Digital...register()" given below the Registration Method Definition) of the registration method definition 2sf20.

At a server registration code generation step 532, the system generates in the registration method definition 2sf20, a code for registration of the server object 20 (refer to Fig. 1) in the routing information table (refer to Fig. 1) with use of the interface name ("DigitalLibrary") as an argument.

Next, the system analyzes the method definition 41 of the interface definition file 4 (refer to Fig. 6) sequentially, repeats the subsequent operations on the basis of a judgement result of a step 533 of judging the presence or absence of a method definition, and in the absence of a definition, terminates its operation at an end step 536.

At a message handler declaration code generation step 534, the system generates the messaging handler declarations 2sf11 and 2sf12 as well as prototype declaration parts (corresponding to "void Digital....keyword" given below Message Handler Definitions 1 and 2) of the messaging handler definitions 2sf21 and 2sf22.

Described in a body of the message handler definition are the original operations of services provided by the server object. The original operations are described by the programmer of the server object.

At a message handler registration code generation step 535, the system generates in registration method definition 2sf20, a code for registering in a message communication library 21 (refer to Fig. 1) a character string corresponding to a conversion of parameters specified in the message description 410 into a standard representation based on predetermined rules as well as a message handler as arguments.

Fig. 9 is a structure of an example of the routing program compiled and generated by the IDL compiler from the interface definition file 4 shown in Fig. 6. The generated routing program 1r is described in source level language. Even in this example, C++ is used as an example of the programming language after the compilation.

The routing program 1r is generated by compiling a routing program header file 1rf1 and a routing program source file 1rf2.

The routing program header file 1rf1 has an interface name variable declaration (corresponding to "static.....InterfaceName" given below the Routing Program Class Declaration, a queuing option variable declaration 1rf11 compiled from the queueing option description 4100 (see Fig. 6), and a matching method declaration 1rf12 to be called by a router 12 (see Fig. 1).

The routing program source file 1rf2 has an interface name variable declaration (corresponding to "static....."DigitalLibrary"" given below the Routing Program Class Definition), an external method declaration 1rf20 compiled from the native method description 40, a queuing option variable definition 1rf21 and a matching method definition 1rf22.

Shown in Fig. 5 is a flowchart for explaining the operation of a routing program generation process 5400 by the routing program generator 54.

The routing program generation process 5400 is called by the parser 51.

At an interface name variable code generation step 5401, the system generates a code for initialization of an interface name variable (corresponding to the above interface name variable definition).

Next, the system judges at a judgement step 5402 the presence or absence of an external method declaration. In the presence of an external method declaration, the system generates al an external method declaration code generation step 5403 the external method declaration 1rf20 compiled from the native method description 40.

Regardless of the judgement results of the judgement step 5402, the system judges at a judgement step 5404 the presence or absence of a queueing option description. In the presence of a queueing option description, the system generates the queuing option variable declaration 1rf11 compiled from the queueing option description 4100 and the queuing option variable definition 1rf21 at a step 5405 of generating a queueing option variable designation value setting code.

Regardless of the judgement result of the step 5402, the system generates at a step 5406 of generating a matching method declaration code the matching method declaration 1rf12 and a prototype declaration part (corresponding to "boolean....message" given below the Matching Method Definition) of the matching method definition 1rf22.

Next, at a message pattern comparison code generating step 5407, the system generates in the matching method definition 1rf22, a code for computing a character string corresponding to a conversion of a parameter of the message pattern specified in the message description 410 into a standard representation based on predetermined rules with a message as an argument of the matching method definition 1rf22. The system generates a code for returning a 'false' indicative of a failure when a coincidence therebetween is not found.

Finally, the system judges the presence or absence

of a matching option description at a judgement step 5408 and, in some cases, generates in the matching method definition 1rf22 a code corresponding to a conversion of the matching option description 411 at a step 5409 of generating a matching option description execute code. The routing program in the presence of the matching description is called a custom routing program.

Regardless of the judgement result at the judgement step, the system terminates the operation of the routing program generator 54 at an end step 5410.

Fig. 10 shows a structure of a message queue.

A send message queue 1m1, a receive message queue 1m2, a transferring send message queue 1m3, a temporary keep message queue 1m4, a send message queue 2m1, a receive message queue 2m2, a transferring send message queue 2m3 and a temporary keep message queue 2m4, shown in Fig. 1 have different use purposes but all have such a message queue structure as shown in Fig. 10.

The send message queue 1m1 includes a message queue ID 1m10 for uniquely identifying a message queue and messages 1m11 and 1m12.

The message 1m11 in turn has a send destination message queue ID 1m110 indicative of send destination objects, a send origination message queue ID list 1m111 indicative of send origination objects, an interface name 1m112 of a service which the send origination object is to use, a message body data 1m113 indicative of a body of the message, and queuing timeout time 1m114.

When consideration is paid to a case of such message integration as will be described later, there exist a plurality of such send origination objects, to which end the send origination message queue ID list 1m111 has a message queue ID 1m1111 and a message queue ID 1m1112.

Fig. 11 shows a structure of a message queue information table 30 (see Fig. 1).

The message queue information table 30 includes message queue information items 301 and 302.

The message queue information item 301 has a message queue ID 3010 and a computer address 3011 indicative of an address of a computer having the message queue present therein.

Fig. 12 shows a structure of a routing information table 31 (see Fig. 1).

The routing information table 31 includes interface information items 311 and 312 corresponding to services provided by the server object.

The interface information item 311 has an interface name 3110, a routing program designation data 3111 indicative of a routing program which the router uses for determining a message transfer destination, and a server object table 3112 indicative of the server object 20 supporting the interface.

In the present invention, a plurality of the server objects 20 can be present to support a single interface, to which end the server object table 3112 has message queue IDs 31121 and 31122 of the receive message queues 2m2 of the server objects 20.

Shown in Fig. 13 is a flowchart for explaining the operation of a client stub process 1ca0 by the client stub 1c.

The client stub process 1ca0 is called by the client object 10.

At a message string generation step 1ca1, the system generates a message string corresponding to a method called by the client object 10.

At a message pattern parameter setting step 1ca2, the system sets an argument passed to the method for the message string as a parameter.

At a step 1ca3 of setting the interface name of the message and the message queue ID, the system sets the message queue ID 1m10 of the client object for the message queue ID 1m1111, sets the interface name for designation of a send destination of the message for the interface name 1m112, and sets the message string for the message body data 1m113 to thereby prepare the message 1m11.

Finally, at a step 1ca4 of storing a send message queue of the message, the system sets the prepared message 1m11 for the message queue 1m1 and then terminates its operation at an end step 1ca5.

Referring to Fig. 14, there is shown a flowchart for explaining the operation of a server skeleton process 2sa0 by the server skeleton 2s.

The server skeleton process 2sa0 is called by the message communication library 21.

At a step 2sa1 of registering the routing information table of the interface name and message queue ID, the system searches the routing information table 31 for the interface name which the server object 20 supports on the basis of such an interface name 3110 as the interface information item 311 within the table, and when finding the interface name, the system registers the receive message queue 2m2 of the server object 20 in the server object table 3112 as the message queue ID 31121.

At a step 2sa2 of registering the message pattern and message handler, the system registers in a message communication library the messaging handler definition 2sf21 and a message pattern for starting the message handler thereof.

At a step 2sa3 of acquiring a message from the receive message queue, the system extracts the message 1m11 from the receive message queue 2m2.

The system next compares the message body data 1m113 of the message 1m11 with the message pattern to judge the presence or absence of the message handler at a step 2sa4, and in the case of the absence of the message handler, the system terminates its operation at an error end step 2sa5.

In the presence of the message handler, the system calls the message handler with use of the parameter extracted from the message as an argument at a mes-

sage handler starting step 2sa6.

Thereafter, the system repeats at a step 2sa3 the operation of acquiring the message from the receive message queue.

Fig. 15 shows a flowchart for explaining the operation of a message routing process by the routing program 1r.

First of all, at a step 1ra01 of acquiring the interface name of a head message of a send message queue, the routing program 1r selects one message 1m11 to be delivered and acquires its interface name 1m112.

At a step 1ra02 of judging the presence or absence of the acquired interface name 1m112 in the routing information table, the system searches the routing information table 31 for the interface name 1m112 on the basis of such an interface name 3110 as the interface information item 311 within the table.

If failing to find the interface name, at a step 1ra03 of storing an error message in the corresponding send queue, the system stores the error message in the receive message queue 1m2 and goes to the step 1ra01 to acquire the interface name of the head message of a send message queue.

If finding the interface name 3110, at a step 1ra04 of calling the matching method for the routing program, the system calls the matching method 1rf22 of the routing program 1r specified by the routing program designation data 3111 with use of the message body data 1m113 as an argument.

On the basis of the called result, the system determines at a step 1ra05 of judging whether or not the matching is successful, the system determines whether or not the determination of a delivery destination is successful.

If failing to determine the delivery destination, the system goes to the step 1ra03 to store the error message in the corresponding receive queue.

If finding the delivery destination, at a step 1ra06 of storing a message in a temporary keep message queue, the system stores the message 1m11 in the temporary keep message queue 1m4.

At this time, the system stores the message queue ID 31121 of the server object table 3112 in the send destination message queue ID 1m110 within the message 1m11, and stores in the queuing timeout time 1m114 a time corresponding to an addition of the time-out time specified in the queuing option variable definition 1rf21 to the storage time.

Further, at a step 1ra07 of judging the presence or absence of the message queue ID, the system performs the operation of the step 1ra06 of storing the message in the temporary keep message queue when finding the presence of the message queue ID 31122 within the server object table.

In the absence of the message queue ID, the system goes to a step 1ra08 to judge the presence or absence of queuing designation.

The queueing designation, which is a parameter designation relating to the message integration, uses the value of the queuing option variable definition 1rf21.

The definition value is 0 by default, in which case the system regards it as no queuing designation and performs no message integration and goes to a step 1ra11 to judge the presence or absence of messages.

In the presence of the queueing designation, the system judges whether or not the number of messages reaches its maximum.

The number of message queue IDs within the send origination message queue ID list 1m111 of the temporary keep message queue 1m4 corresponds to the number of integrated messages. Thus, when the ID number is larger than the designated maximum number, the system goes to the step 1ra11 to judge the presence or absence of messages.

In the case of absence of messages, the system judges whether or not the timeout time expired.

When the current time goes behind the timeout time 1m114, the system goes to the step 1ra11 to judge the presence or absence of messages.

In the case of the message absence, the system goes to the step 1ra01 to acquire the interface of the head message of a send message queue for processing of the next message.

At the step 1ra11 of judging the presence or absence of messages, the system searches for the temporary keep message queue 1m4 and, when the time-out time 1m114 of the message 1m11 expires, the system goes to a step ira12 to judge whether or not the computer addresses are the same.

If there is no message whose time is behind the timeout time, the system goes to the step 1ra01 to acquire the interface of the head message of a send message queue for processing of the next message.

At the step 1ra12 of judging whether or not the computer addresses are the same, the system compares the send destination message queue ID 1m110 of the message 1m11 with the message queue ID 3010 stored in the message queue information 301 and 302 of the message queue information table 30, and acquires the computer address 3011 of the coincided message queue information 301.

When the computer address 3011 is not the current computer address, the system goes to a step 1ra13 to transfer to the transferring send message queue of another computer, and for example, the system transfers the message 1m11 to the transferring send message queue 2m3 of the computer 2.

Regardless of the above judgement result, the system proceeds to a step 1ra14 to transfer a message to the next corresponding receive message queue. For example, the router 22 transfers the message 1m11 to the receive message queue 2m2 corresponding to the send destination message queue ID 1m110.

Thereafter, in order to process another message within the temporary keep message queue 1m4, the system repeats the operation of the step 1ra11 to judge

the presence or absence of a message.

The embodiment of the present invention has been explained above. Another embodiment thereof will be explained in detail below.

Fig. 17 shows a structure of a server object 70 for centralizedly managing several programs for services.

The service object 70 includes an interface name 700, the client stub 1c, server skeleton 2s and routing program 1r in the foregoing embodiment.

Shown in Fig. 16 is an arrangement of an inter-object communication system which uses a dynamic service object loader for causing the client object 10, server object 20 and router 22 to load the service object of Fig. 17 as necessary.

In the present arrangement, dynamic object service loaders 6d1, 6d2 and 6d3 and a memory 7m for storing the service object 70 therein are additionally provided in the entire arrangement of the foregoing embodiment.

The dynamic object service loader 6d1, which is called by the client object 10, dynamically loads the service object 70 and passes the client stub 1c to the client object 10.

Thereafter, the client object 10 utilizes the client stub 1c as in the foregoing embodiment.

Even the dynamic object service loader 6d2 and 6d3 also each perform similar operations in the server object 20 and router 22.

The memory 7m may be placed on the computer 6 or 3 if necessary.

The present invention has for example the following six advantages (1) to (6) in accordance with the aforementioned method.

(1) When the client object designates the interface name for its communication, the service can be used without explicitly designating the server object.

(2) The object which provides the service is not always single. When the router delivers a message to a plurality of communication destinations, one-to-multipoint communication can be realized without the programmer of the client object recognizes it.

(3) Even when the object which provides the service is added, deleted or changed, the message can be delivered without causing the router to affect the client object, by the server object causing to suitably update the routing information (relating to the service provided by its own).

(4) When the service provider or the programmer of the server object customizes the routing method for causing the router to determine the communication destination, finer message delivery control can be realized.

(5) When the service provider or the programmer of the server object specifies a method for causing the router to integrate messages having the same contents, the amount of messages can be reduced and thereby the load of the server object can be reduced.

(6) When the client stub, server skeleton and routing program are combined into a single service object so that, at the time of execution, the client object, server object and router will dynamically load the service object respectively; the program modules can be centralizedly managed and application installation can be simplified.

Claims

1. A method for converting an interface definition description in an inter-object communication system wherein a client object (10) operating on either one of a plurality of computers (1,2,3) in a network (8) or on a single computer sends a message to a server object (20) operating on the same or another computer to execute a predetermined processing operation, said method comprising:

   causing an interface definition converting program to convert an interface definition description described in an interface definition language to thereby generate a client stub (1c), a server skeleton (2s) and a routing program (1r,2r),

   said interface definition description including one or more of a method definition description of data necessary for said predetermined processing operation, and a processing result, and a message description specifying a format of the message to be sent in response to said method definition description,

   said client stub being called to cause said client object to send the message,

   said server skeleton including a server registration method for storing, in a routing information memory routing information for specifying the format of said message which said server object itself can process when started and also including a method for describing said predetermined processing operation to be executed when said server object receives the message, and

   said routing program judging whether or not the processing of said server object associated with said routing information of said message is possible through comparison between said message and said routing information.

2. An inter-object communication method wherein a client object (10) operating on either one of a plurality of computers (1,2,3) in a network (8) or on a single computer sends a message to a server object (20) operating on the same or another computer to execute a predetermined processing operation, said method comprising the steps of:

storing a client stub (1c), a server skeleton (2s) and a routing program (1r,2r) generated based on an interface definition description;

when said server object is started, calling a server registration method and storing routing information in a routing information memory (3m) to cause said client object to call said client stub to send the message and to hold said sent message in a send message queue (1m1,2m1);

causing a router (12,22) to extract the message from said send message queue, to determine a send destination object in accordance with said routing program and to add the received message in a receive message queue (1m2,2m2) associated with said send destination object; and

causing said server skeleton to extract the message from said receive message queue and causing said server object to execute said predetermined processing operation of the message extracted by said server skeleton.

3. A method for converting an interface definition description as set forth in claim 1, wherein said routing program adds a queueing option description including a message maximum number designation and a queueing time designation to a message description of said interface definition description, compares said message and said routing information to judge whether or not the processing operation by said server object associated with said routing information of said message is possible and, in the case of presence of said message maximum number designation, performs said queueing option judging operation.

4. An inter-object communication method as set forth in claim 2, wherein said router keeps the message extracted from said send message queue in a temporary keep message queue, and when another message has the same contents as said kept message, the router integrates the messages into a single message, and when the number of messages within the temporary keep message queue exceeds said message maximum number or when said queueing time elapses after said message is queued, the router adds said message to a receive message queue associated with said send destination object.

5. A method for converting an interface definition description as set forth in claim 1, wherein, in place of said routing program, a custom routing program is generated which compares said message and said routing information to judge whether or not the processing operation by said server object associated with said routing information of said message is possible and performs said matching option judging operation.

6. An inter-object communication method as set forth in claim 2, further comprising the steps of:

combining said client stub, said server skeleton and said routing program into a single server object and storing said service object in a service object memory;

before said client object sends a message, extracting said server object from said service object memory to call said client stub;

before said router determines said send destination object, extracting said server object from said service object memory to call said routing program; and

at the time of starting said server object, extracting said server object from said service object memory to call said server skeleton.

FIG.1

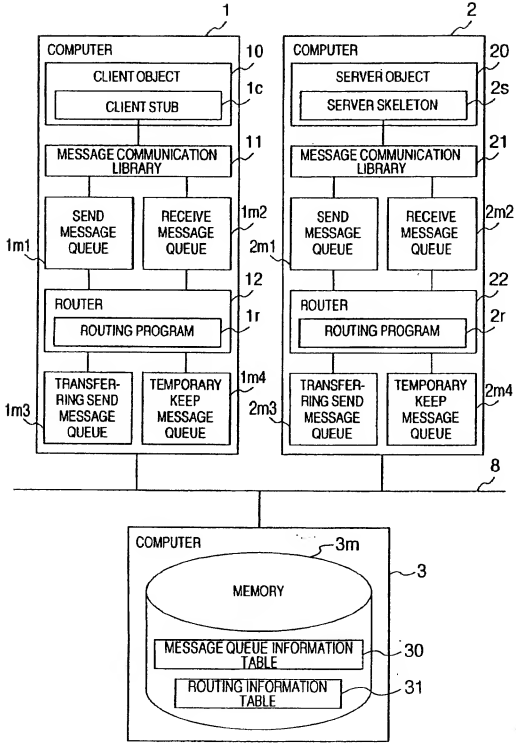

1
2
COMPUTER
10
CLIENT OBJECT
1c
CLIENT STUB
11
MESSAGE COMMUNICATION LIBRARY
SEND MESSAGE QUEUE
RECEIVE MESSAGE QUEUE
1m2
1m1
2m1
12
ROUTER
1r
ROUTING PROGRAM
TRANSFER-RING SEND MESSAGE QUEUE
TEMPORARY KEEP MESSAGE QUEUE
1m4
1m3
2m3
COMPUTER
20
SERVER OBJECT
2s
SERVER SKELETON
21
MESSAGE COMMUNICATION LIBRARY
SEND MESSAGE QUEUE
RECEIVE MESSAGE QUEUE
2m2
22
ROUTER
2r
ROUTING PROGRAM
TRANSFER-RING SEND MESSAGE QUEUE
TEMPORARY KEEP MESSAGE QUEUE
2m4
8
COMPUTER
3m
3
MEMORY
MESSAGE QUEUE INFORMATION TABLE
30
ROUTING INFORMATION TABLE
31

FIG.2

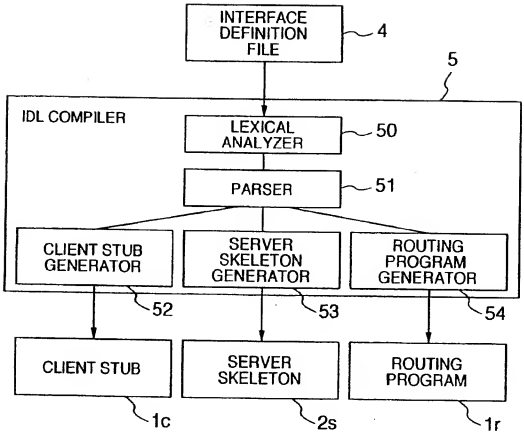

INTERFACE DEFINITION FILE
4
5
IDL COMPILER
LEXICAL ANALYZER
50
PARSER
51
CLIENT STUB GENERATOR
SERVER SKELETON GENERATOR
ROUTING PROGRAM GENERATOR
52
53
54
CLIENT STUB
SERVER SKELETON
ROUTING PROGRAM
1c
2s
1r

FIG.3

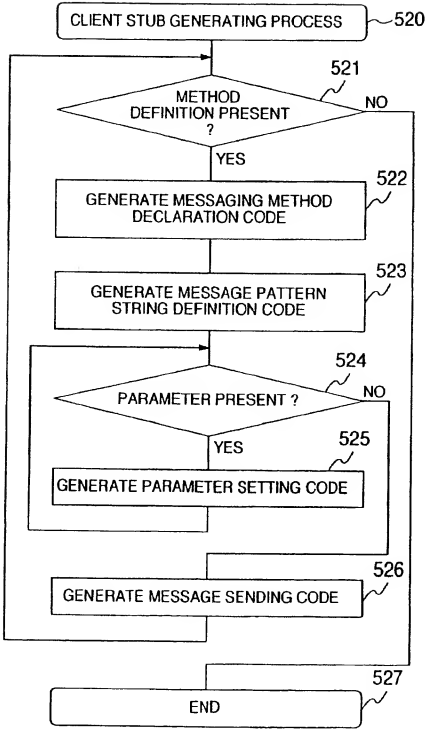

CLIENT STUB GENERATING PROCESS
520
521
METHOD DEFINITION PRESENT ?
NO
YES
522
GENERATE MESSAGING METHOD DECLARATION CODE
523
GENERATE MESSAGE PATTERN STRING DEFINITION CODE
524
PARAMETER PRESENT ?
NO
YES
525
GENERATE PARAMETER SETTING CODE
526
GENERATE MESSAGE SENDING CODE
527
END

FIG.4

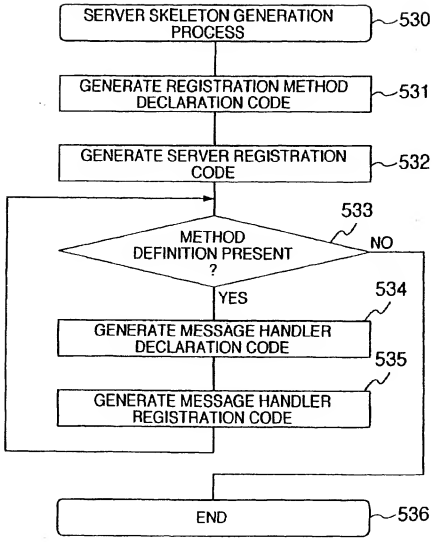

SERVER SKELETON GENERATION PROCESS
530
GENERATE REGISTRATION METHOD DECLARATION CODE
531
GENERATE SERVER REGISTRATION CODE
532
533
METHOD DEFINITION PRESENT ?
NO
YES
534
GENERATE MESSAGE HANDLER DECLARATION CODE
535
GENERATE MESSAGE HANDLER REGISTRATION CODE
END
536

FIG.5

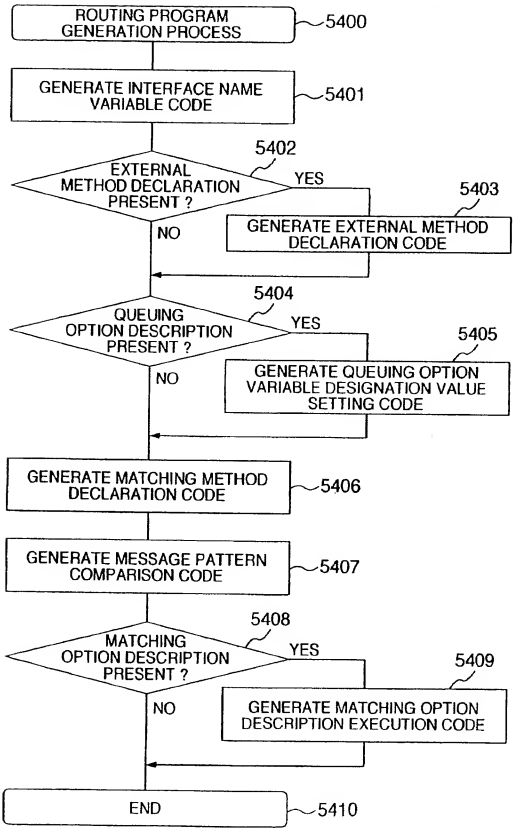

ROUTING PROGRAM GENERATION PROCESS
5400
GENERATE INTERFACE NAME VARIABLE CODE
5401
5402
EXTERNAL METHOD DECLARATION PRESENT ?
YES
NO
5403
GENERATE EXTERNAL METHOD DECLARATION CODE
5404
QUEUING OPTION DESCRIPTION PRESENT ?
YES
NO
5405
GENERATE QUEUING OPTION VARIABLE DESIGNATION VALUE SETTING CODE
GENERATE MATCHING METHOD DECLARATION CODE
5406
GENERATE MESSAGE PATTERN COMPARISON CODE
5407
5408
MATCHING OPTION DESCRIPTION PRESENT ?
YES
NO
5409
GENERATE MATCHING OPTION DESCRIPTION EXECUTION CODE
END
5410

FIG.6

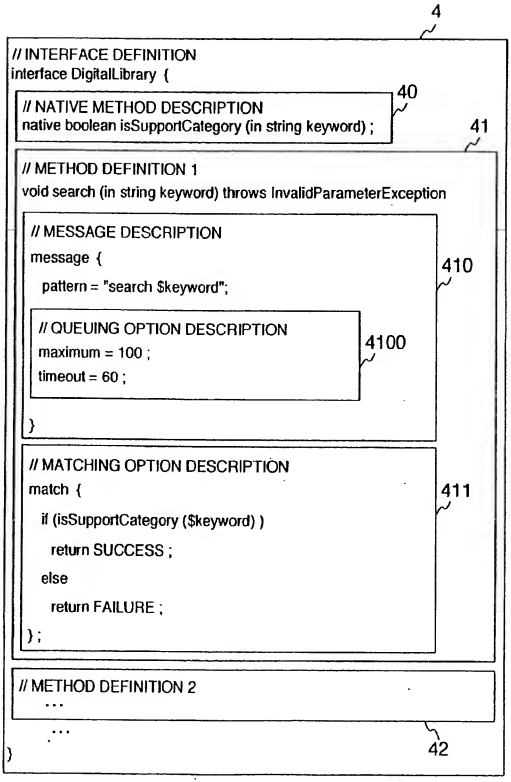

4
// INTERFACE DEFINITION
interface DigitalLibrary {
// NATIVE METHOD DESCRIPTION
native boolean isSupportCategory (in string keyword) ;
40
41
// METHOD DEFINITION 1
void search (in string keyword) throws InvalidParameterException
// MESSAGE DESCRIPTION
message {
pattern = "search $keyword";
410
// QUEUING OPTION DESCRIPTION
maximum = 100 ;
timeout = 60 ;
4100
}
// MATCHING OPTION DESCRIPTION
match {
if (isSupportCategory ($keyword))
return SUCCESS ;
else
return FAILURE ;
} ;
411
// METHOD DEFINITION 2
...
...
}
42

FIG.7

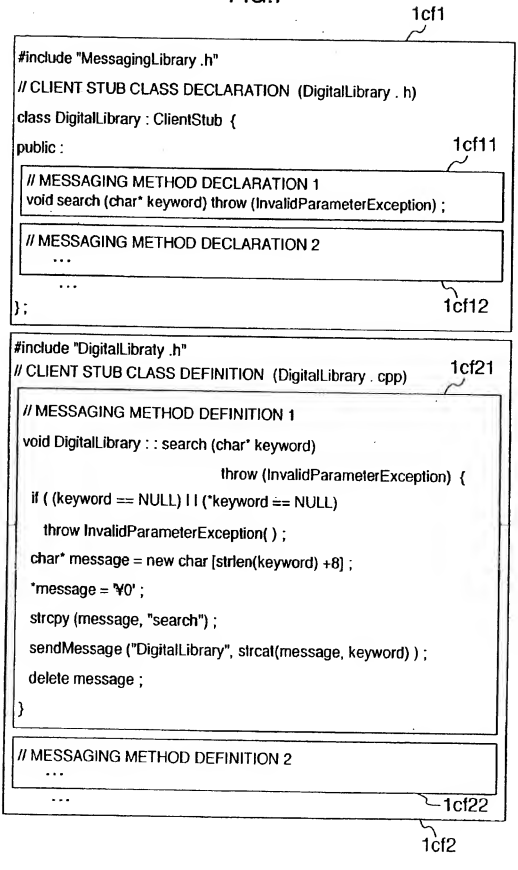

1cf1
#include "MessagingLibrary .h"
// CLIENT STUB CLASS DECLARATION (DigitalLibrary . h)
class DigitalLibrary : ClientStub {
public :
1cf11
// MESSAGING METHOD DECLARATION 1
void search (char* keyword) throw (InvalidParameterException) ;
// MESSAGING METHOD DECLARATION 2
...
...
} ;
1cf12
#include "DigitalLibraty .h"
// CLIENT STUB CLASS DEFINITION (DigitalLibrary . cpp)
1cf21
// MESSAGING METHOD DEFINITION 1
void DigitalLibrary : : search (char* keyword)
throw (InvalidParameterException) {
if ((keyword == NULL) ! ! (*keyword == NULL)
throw InvalidParameterException( ) ;
char* message = new char [strlen(keyword) +8] ;
*message = '¥0' ;
strcpy (message, "search") ;
sendMessage ("DigitalLibrary", strcat(message, keyword)) ;
delete message ;
}
// MESSAGING METHOD DEFINITION 2
...
...
1cf22
1cf2

FIG.8

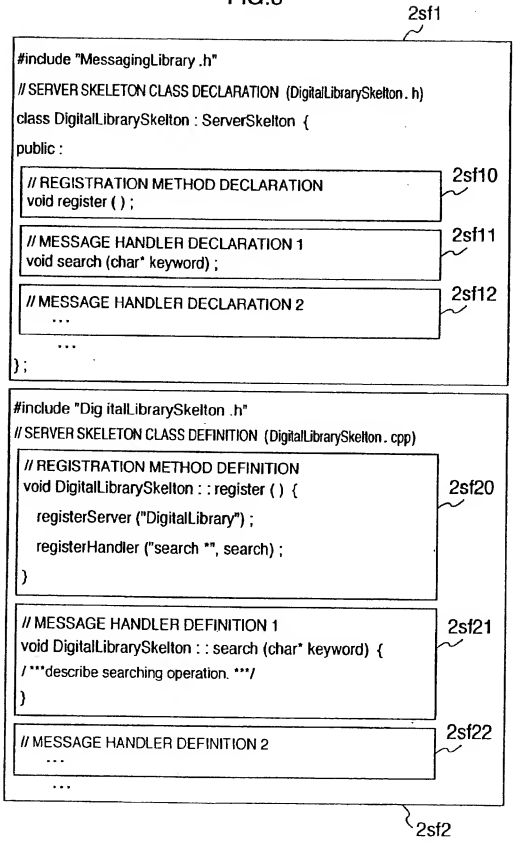

2sf1
#include "MessagingLibrary .h"
// SERVER SKELETON CLASS DECLARATION (DigitalLibrarySkelton. h)
class DigitalLibrarySkelton : ServerSkelton {
public :
// REGISTRATION METHOD DECLARATION
void register ( ) ;
2sf10
// MESSAGE HANDLER DECLARATION 1
void search (char* keyword) ;
2sf11
// MESSAGE HANDLER DECLARATION 2
...
...
2sf12
) ;
#include "Dig italLibrarySkelton .h"
// SERVER SKELETON CLASS DEFINITION (DigitalLibrarySkelton. cpp)
// REGISTRATION METHOD DEFINITION
void DigitalLibrarySkelton : : register ( ) {
registerServer ("DigitalLibrary") ;
registerHandler ("search "", search) ;
}
2sf20
// MESSAGE HANDLER DEFINITION 1
void DigitalLibrarySkelton : : search (char* keyword) {
/ ***describe searching operation. ***/
}
2sf21
// MESSAGE HANDLER DEFINITION 2
...
...
2sf22
2sf2

FIG.9

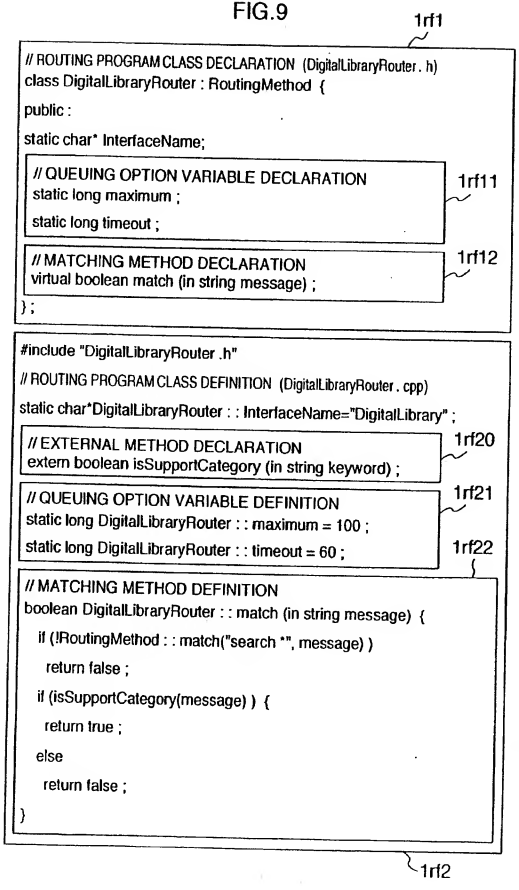

1rf1
// ROUTING PROGRAM CLASS DECLARATION (DigitalLibraryRouter. h)
class DigitalLibraryRouter : RoutingMethod {
public :
static char* InterfaceName;
// QUEUING OPTION VARIABLE DECLARATION
static long maximum ;
static long timeout ;
1rf11
// MATCHING METHOD DECLARATION
virtual boolean match (in string message) ;
1rf12
} ;
#include "DigitalLibraryRouter .h"
// ROUTING PROGRAM CLASS DEFINITION (DigitalLibraryRouter. cpp)
static char*DigitalLibraryRouter : : InterfaceName="DigitalLibrary" ;
// EXTERNAL METHOD DECLARATION
extern boolean isSupportCategory (in string keyword) ;
1rf20
// QUEUING OPTION VARIABLE DEFINITION
static long DigitalLibraryRouter : : maximum = 100 ;
static long DigitalLibraryRouter : : timeout = 60 ;
1rf21
1rf22
// MATCHING METHOD DEFINITION
boolean DigitalLibraryRouter : : match (in string message) {
if (!RoutingMethod : : match("search *", message))
return false ;
if (isSupportCategory(message)) {
return true ;
else
return false ;
}
1rf2

FIG.10

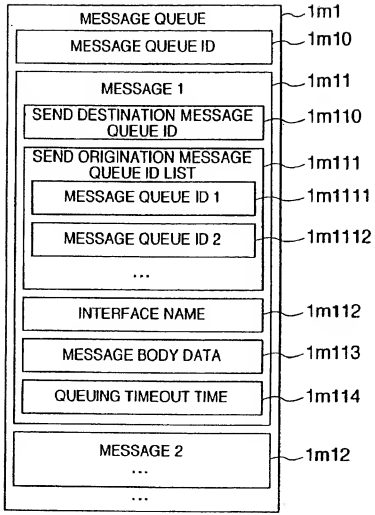

MESSAGE QUEUE
1m1
MESSAGE QUEUE ID
1m10
MESSAGE 1
1m11
SEND DESTINATION MESSAGE QUEUE ID
1m110
SEND ORIGINATION MESSAGE QUEUE ID LIST
1m111
MESSAGE QUEUE ID 1
1m1111
MESSAGE QUEUE ID 2
1m1112
...
INTERFACE NAME
1m112
MESSAGE BODY DATA
1m113
QUEUING TIMEOUT TIME
1m114
MESSAGE 2
...
1m12
...

FIG.11

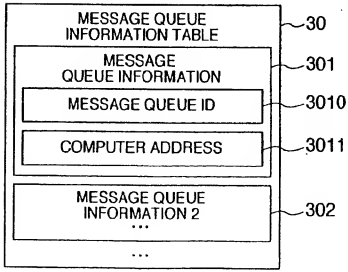

MESSAGE QUEUE INFORMATION TABLE
30
MESSAGE QUEUE INFORMATION
301
MESSAGE QUEUE ID
3010
COMPUTER ADDRESS
3011
MESSAGE QUEUE INFORMATION 2
...
302
...

FIG.12

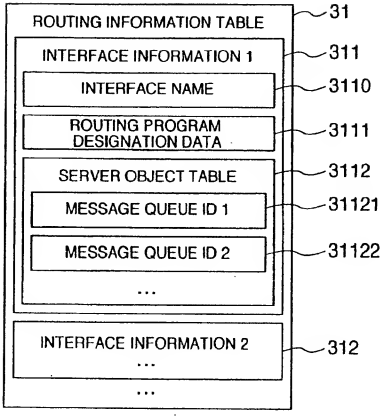

ROUTING INFORMATION TABLE
31
INTERFACE INFORMATION 1
311
INTERFACE NAME
3110
ROUTING PROGRAM DESIGNATION DATA
3111
SERVER OBJECT TABLE
3112
MESSAGE QUEUE ID 1
31121
MESSAGE QUEUE ID 2
31122
...
INTERFACE INFORMATION 2
312
...
...

FIG.13

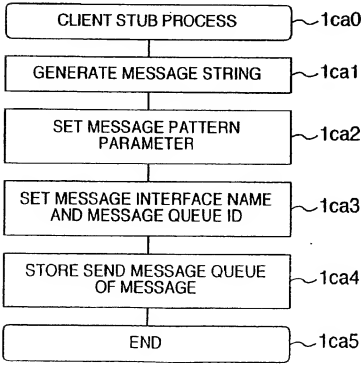

CLIENT STUB PROCESS
1ca0
GENERATE MESSAGE STRING
1ca1
SET MESSAGE PATTERN PARAMETER
1ca2
SET MESSAGE INTERFACE NAME AND MESSAGE QUEUE ID
1ca3
STORE SEND MESSAGE QUEUE OF MESSAGE
1ca4
END
1ca5

FIG.14

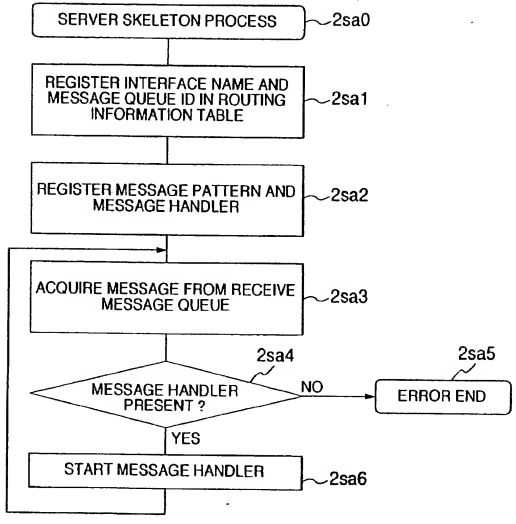

SERVER SKELETON PROCESS
2sa0
REGISTER INTERFACE NAME AND MESSAGE QUEUE ID IN ROUTING INFORMATION TABLE
2sa1
REGISTER MESSAGE PATTERN AND MESSAGE HANDLER
2sa2
ACQUIRE MESSAGE FROM RECEIVE MESSAGE QUEUE
2sa3
2sa4
MESSAGE HANDLER PRESENT ?
NO
2sa5
ERROR END
YES
START MESSAGE HANDLER
2sa6

FIG.17

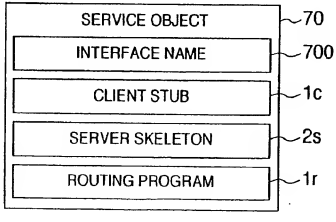

SERVICE OBJECT
70
INTERFACE NAME
700
CLIENT STUB
1c
SERVER SKELETON
2s
ROUTING PROGRAM
1r

FIG.15

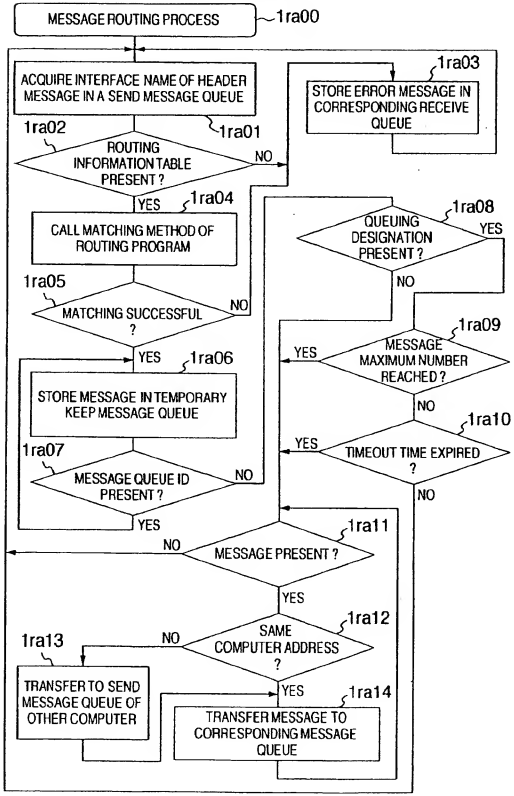
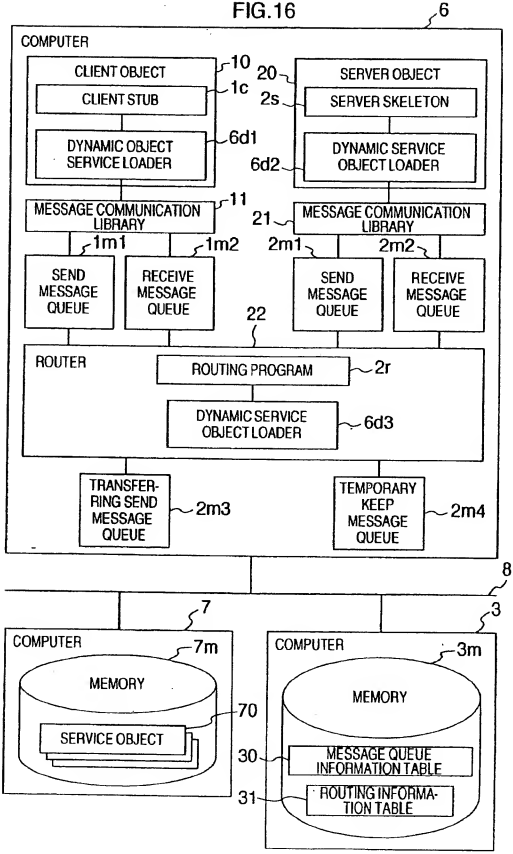

MESSAGE ROUTING PROCESS
1ra00
ACQUIRE INTERFACE NAME OF HEADER MESSAGE IN A SEND MESSAGE QUEUE
1ra01
1ra02
ROUTING INFORMATION TABLE PRESENT ?
NO
YES
1ra03
STORE ERROR MESSAGE IN CORRESPONDING RECEIVE QUEUE
1ra04
CALL MATCHING METHOD OF ROUTING PROGRAM
1ra05
MATCHING SUCCESSFUL ?
NO
YES
1ra06
STORE MESSAGE IN TEMPORARY KEEP MESSAGE QUEUE
1ra07
MESSAGE QUEUE ID PRESENT ?
NO
YES
1ra08
QUEUING DESIGNATION PRESENT ?
YES
NO
1ra09
MESSAGE MAXIMUM NUMBER REACHED ?
YES
NO
1ra10
TIMEOUT TIME EXPIRED ?
YES
NO
1ra11
MESSAGE PRESENT ?
NO
YES
1ra12
SAME COMPUTER ADDRESS ?
NO
YES
1ra13
TRANSFER TO SEND MESSAGE QUEUE OF OTHER COMPUTER
1ra14
TRANSFER MESSAGE TO CORRESPONDING MESSAGE QUEUE

FIG.16


6
COMPUTER
CLIENT OBJECT
10
CLIENT STUB
1c
DYNAMIC OBJECT SERVICE LOADER
6d1
20
SERVER OBJECT
2s
SERVER SKELETON
DYNAMIC SERVICE OBJECT LOADER
6d2
MESSAGE COMMUNICATION LIBRARY
11
21
MESSAGE COMMUNICATION LIBRARY
1m1
1m2
2m1
2m2
SEND MESSAGE QUEUE
RECEIVE MESSAGE QUEUE
SEND MESSAGE QUEUE
RECEIVE MESSAGE QUEUE
22
ROUTER
ROUTING PROGRAM
2r
DYNAMIC SERVICE OBJECT LOADER
6d3
TRANSFER-RING SEND MESSAGE QUEUE
2m3
TEMPORARY KEEP MESSAGE QUEUE
2m4
8
7
3
COMPUTER
7m
MEMORY
70
SERVICE OBJECT
COMPUTER
3m
MEMORY
30
MESSAGE QUEUE INFORMATION TABLE
31
ROUTING INFORMATION TABLE

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 98 10 2297

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.6)
A	KOON-YUI POON: "INSIDE A TRADER IN GLOBAL TRADING" COMPUTER COMMUNICATIONS, vol. 18, no. 4, 1 April 1995, pages 227-246, XP000498295 * page 244, left-hand column, line 22 - page 246, left-hand column, line 20; figures 8,9 *	1,2	G06F9/46
A	EP 0 592 091 A (IBM) 13 April 1994 * abstract; figures 4A,4B *	1,2	
			TECHNICAL FIELDS SEARCHED (Int.Cl.6)
			G06F

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	3 July 1998	Kingma, Y

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)

Requested Patent: FR2748620A1

Title:

ARBITRATION METHOD FOR QUEUED DATA AWAITING TRANSMISSION OVER DIGITAL NETWORK ;

Abstracted Patent: FR2748620 ;

Publication Date: 1997-11-14 ;

Inventor(s): COTTIGNIES VINCENT;; MOUEN MAKOUA DAVID ;

Applicant(s): THOMSON CSF (FR) ;

Application Number: FR19960005845 19960510 ;

Priority Number(s): FR19960005845 19960510 ;

IPC Classification: H04J3/22; H04L12/26 ;

Equivalents: ;

ABSTRACT:

The method involves determining the number of data queues. Each data queue is associated with a data priority determined by a circuit in an input interface. The circuit includes a counter (41) and an eligibility table (42). This table consists of a number of registers equal to the number of node output ports of the network. A binary signal is used to indicate the eligibility of each waiting queue, with a logic 0 indicating that queue is not eligible and a logic 1 indicating that it is. A queue is determined to be eligible if it is not empty and if the destination is not congested. A random number generator is used to determine which queued data is to be transmitted when there are a number of eligible queues with the same priority.

(19) RÉPUBLIQUE FRANÇAISE

INSTITUT NATIONAL
DE LA PROPRIÉTÉ INDUSTRIELLE

PARIS

(11) N° de publicati n : **2 748 620**
(à n'utiliser que pour les commandes de reproduction)

(21) N° d'enregistrement national : **96 05845**

(51) Int Cl⁶ : H 04 J 3/22, H 04 L 12/26

(12) DEMANDE DE BREVET D'INVENTION A1

(22) Date de dépôt : 10.05.96.

(30) Priorité :

(43) Date de la mise à disposition du public de la demande : 14.11.97 Bulletin 97/46.

(56) Liste des documents cités dans le rapport de recherche préliminaire : *Se reporter à la fin du présent fascicule.*

(60) Références à d'autres documents nationaux apparentés :

(71) Demandeur(s) : *THOMSON CSF SOCIETE ANONYME* — FR.

(72) Inventeur(s) : COTTIGNIES VINCENT et MOUEN MAKOUA DAVID.

(73) Titulaire(s) : .

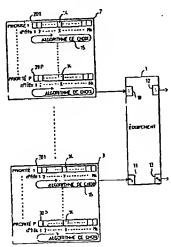
(74) Mandataire : THOMSON CSF.

(54) PROCEDE ET DISPOSITIF D'ARBITRAGE ENTRE DES FILES D'ATTENTE DANS UN RESEAU DE TRANSMISSION NUMERIQUE.

(57) L'invention concerne un procédé et dispositif d'arbitrage entre des files d'attente dans un réseau de transmission numérique.

Une file d'attente ayant un niveau de priorité attribué, un aléa est introduit dans la sélection des files d'attente de même priorité.

Application: Equipement de brassage et de commutation de données numériques notamment dans un réseau fonctionnant en mode de transmission asynchrone (ATM) par exemple.



FR 2 748 620 - A1

La présente invention concerne un procédé et dispositif d'arbitrage entre des files d'attente dans un réseau de transmission numérique. Elle s'applique notamment aux équipements de brassage et de commutation de données numériques composant un réseau fonctionnant en mode de transmission asynchrone dit ATM, ce dernier terme étant dérivé de l'expression anglo-saxonne "Asynchronous Transfer Mode". Plus généralement, l'invention s'applique à tous les réseaux de transmission de données numériques où il est nécessaire pour aiguiller ces données d'arbitrer entre files d'attentes en différents points du réseau.

Les systèmes de télécommunication font notamment de plus en plus appel au mode de transfert asynchrone ATM. Cette technique de transmission permet de véhiculer des informations numériques de natures et de débits aussi variés qu'irréguliers. L'information est transportée à travers un réseau pouvant présenter diverses topologies, par exemple en maillage, en anneau ou en étoile. Chaque noeud du réseau, appelé commutateur ATM, est relié à un noeud adjacent par une artère de transmission acceptant tous types de technologies. La transmission peut donc se faire par exemple par câble, par faisceau hertzien ou par fibre optique.

Le principe de la technique ATM est de segmenter les données issues des divers sources en blocs de longueur fixe, et d'ajouter à un bloc un en-tête pour former une cellule ATM. Les cellules ATM provenant de diverses sources sont alors multiplexées et transmises de façon asynchrone sur l'artère de transmission.

Au sein d'équipements ATM, à un instant donné, plusieurs cellules peuvent se disputer l'accès à une même ressource. La mise en file d'attente des cellules concurrentes est alors nécessaire et il faut par la suite choisir selon quel ordre les files d'attente seront servies.

Dans un équipement, par exemple un commutateur ATM, possédant Ne ports d'entrée et Ns ports de sortie, il est nécessaire de mettre en file d'attente les cellules reçues ou à émettre. Au niveau d'un port d'entrée, les cellules sont stockées dans différents registres de type FIFO selon des critères précis, par exemple selon la destination, le chemin ou le canal virtuel, le port de sortie de l'équipement suivant ou la priorité. Un registre de type FIFO est un registre connu de l'homme du métier où les bits

sortent selon leur ordre d'entrée dans le registre, FIFO étant dérivé de l'expression anglo-saxonne "First in, First out".

Selon l'art antérieur, chaque port possède un algorithme d'arbitrage par niveau de priorité qui permet de choisir la file d'attente à servir parmi toutes celles qui ont la même priorité. Ce mécanisme se réfère à une table d'éligibilité qui contient des informations binaires qui selon leurs valeurs autorisent ou non de servir un client ou un ensemble de clients. Par exemple, cette table d'éligibilité interdit de servir les files d'attente dont les cellules sont destinées à des ports de sortie congestionnés. Un des algorithmes d'arbitrage le plus utilisé est connu sous le terme de "Priorité Tournante" ou de "Round Robin". L'algorithme opère indépendamment d'un port d'entrée à l'autre.

Les mécanismes d'arbitrage actuels introduisent plusieurs inconvénients, en particulier ils peuvent entraîner une dégradation des performances. Le rendement est ainsi mauvais dans la mesure où il apparaît des congestions à l'intérieur des commutateurs. Il en résulte alors pour l'utilisateur final une augmentation des temps de réponse. A terme, l'usager est donc mal servi.

Le but de l'invention est de pallier les inconvénients précités pour permettre notamment un écoulement efficace et rapide des informations au niveau de chaque noeud d'un réseau par exemple asynchrone, et donc améliorer le service final pour l'usager.

A cet effet, l'invention a pour objet un procédé d'arbitrage entre des files d'attente au niveau d'un noeud d'un réseau de transmission numérique, caractérisé en ce que chaque file d'attente ayant un niveau de priorité attribué, il introduit un aléa dans la sélection des files d'attente de même priorité.

L'invention a également pour objet un dispositif pour la mise en oeuvre du procédé. L'invention a pour principaux avantages qu'elle est simple à mettre en oeuvre et qu'elle est économique.

D'autres caractéristiques et avantages de l'invention apparaîtront à l'aide de la description qui suit faite en regard de dessins annexés qui représentent :

- la figure 1, un dispositif d'arbitrage fonctionnant selon l'art antérieur,

- les figures 2a, 2b, 2c des illustrations d'un problème de synchronisation mis en évidence par la Déposante,

- la figure 3, un exemple de mode de réalisation possible pour la mise en oeuvre du procédé selon l'invention.

La figure 1 présente, par un synoptique, un dispositif d'arbitrage entre files d'attente au niveau d'un noeud d'un réseau numérique, selon l'art antérieur. Un équipement 1 est voué à recevoir des données (cellules ATM ou paquets ou trames ...) provenant de Ne ports d'entrée et à les retransmettre sur certains de ses Ns ports de sortie. Les ports de sortie ayant un débit limité, si tous les ports d'entrée présentent des données destinées au même port de sortie, il sera impossible de retransmettre toutes ces cellules en même temps. Les cellules qui n'auront pu être servies seront alors mises en files d'attente dans la FIFO du port de sortie destinataire et retransmises ultérieurement. Lorsque ces FIFO de sortie atteignent un certain seuil, on dit qu'il apparaît une congestion au sein de l'équipement. Afin de ne pas aller plus avant dans la congestion, elle doit être notifiée à tous les ports d'entrée par l'intermédiaire d'un contrôle de flux entre sorties et entrées et/ou pour la mise à jour d'une table d'éligibilité. Ainsi pour chaque port d'entrée, un arbitre est nécessaire pour choisir les files d'attente à servir parmi celles qui ne sont pas susceptibles d'aggraver la congestion. Un algorithme de choix est alors nécessaire ; c'est l'algorithme "Round Robin", appelé encore algorithme à "Priorité Tournante".

Des cellules arrivent à un noeud, concrétisé par un équipement 1. Cet équipement est par exemple un commutateur ATM. Sa fonction est notamment d'aiguiller les cellules arrivant sur ses ports d'entrée 10,11 vers ses ports de sortie 12, 13. L'équipement 1 comporte par exemple Ne ports d'entrée 10, 11. Une première interface d'entrée 2 est reliée au premier port d'entrée 10 de l'équipement 1. Une Ne^{ième} interface 3 est reliée au Ne ^{ième} port d'entrée 11. Une interface, par exemple la première 2, comporte des circuits 201, 20P constitués de registres 14 de type FIFO, un premier circuit 201 contenant toutes les files d'attente de priorité 1, la priorité la plus élevée, un pième circuit 20P contenant toutes les files d'attente de priorité P, la priorité la moins élevée, et d'autres circuits contenant les files d'attente de priorité intermédiaires. A titre d'exemple, le fonctionnement des interfaces 2, 3 est expliqué relativement aux circuits 201, 301 de priorité 1 de ces

interfaces. Un circuit de priorité 1 sera appelé par la suite priorité 1. Chaque port de sortie du noeud 1 du réseau possède un algorithme d'arbitrage par niveau de priorité.

Une priorité 1 comporte un nombre Ns de registres 14 de type FIFO, ce nombre Ns étant égal au nombre de ports de sortie de l'équipement 1. Chaque registre 14 de la priorité 1 contient une ou plusieurs cellules entrantes. Le rang X du registre 14, encore appelé numéro de FIFO, correspond au numéro du port de sortie prévu pour la cellule entrante. Le numéro X de FIFO précité correspond en fait au numéro de file d'attente de la cellule entrante. Par exemple une cellule ayant le rang X est prévue pour sortir sur le Xième port de sortie du noeud 1 du réseau. En d'autres termes, une FIFO contient une file d'attente de cellules, la FIFO 14 pouvant éventuellement être vide.

A chaque priorité 201, 20P, 301, 30P est associé un algorithme de choix 15 déroulé par exemple par un processeur non représenté. Les algorithmes d'arbitrage généralement utilisés sont connus sous l'appellation "Round Robin". L'algorithme d'arbitrage 15 associé à la priorité 1 sélectionne un rang X après avoir sélectionné un rang inférieur, et cela parmi les FIFO qui ont le droit d'émettre une cellule vers la sortie du noeud 1. Les cellules associées sont dites éligibles, par extension on dira que ce sont les rangs ou numéros de FIFO qui sont éligibles. Une table d'éligibilité non représentée est associée aux priorités ou circuits 201, 20P. Une FIFO dont le rang X a été sélectionné émet une cellule vers le port de sortie de rang X de l'équipement 1.

Au tour suivant, l'algorithme sélectionne un rang suivant, X + 1 si celui-ci est éligible, puis le rang X + 2, et ainsi de suite. A chaque FIFO 14 est associée une adresse de la table d'éligibilité pour aller y lire une information binaire sur son état d'éligibilité. L'algorithme d'arbitrage 15 opère donc cycliquement sur tous les rangs ou numéros de FIFO 1, 2, ...X, N_S d'une même priorité, dans l'exemple de fonctionnement expliqué, il s'agit en l'occurrence des registres de priorité 1, donc des files d'attente associées.

La Déposante a mis en évidence des problèmes de synchronisation de tous les algorithmes d'arbitrage de tous les ports d'entrée d'un noeud d'un réseau lorsqu'ils sont soumis à de fortes charges notamment quand toutes les files d'attente, c'est-à-dire toutes les priorités

201, 20P, 301, 308 contiennent des cellules. Le même phénomène de synchronisation peut évidemment se retrouver aussi au niveau des ports de sortie s'ils sont connectés à un autre noeud du réseau, c'est-à-dire que le même principe peut être appliqué entre différents noeuds d'un réseau. Dans ce cas, les cellules sont rangées en FIFO dont le numéro indique le port de sotie de l'équipement suivant. Le contrôle de flux est cette fois transmis entre noeuds.

Deux problèmes de synchronisation ont été mis en évidence par la Déposante.

Une synchronisation de type convergente apparaît lorsque les algorithmes de choix évoluent vers un état de synchronisation où tous les ports, d'entrée ou de sortie, servent la même file d'attente en même temps.

Une synchronisation de type aléatoire se produit lorsqu'une certaine périodicité dans le choix des files d'attente à servir provoque, à un instant t imprévisible, la synchronisation des algorithmes indépendants. A cet instant t, les algorithmes présentent différents déphasages les uns par rapport aux autres, et ils les conserveront tant qu'ils resteront synchronisés. Cela signifie notamment que si à l'instant t, une interface d'entrée sert la file de rang i et une autre interface d'entrée sert la file de rang j, alors l'écart de déphasage |j-i| restera constant tant que les files d'attente restent toutes non vides.

A ces deux problèmes de synchronisation correspondent en fait deux types d'algorithmes basés sur des contrôles de flux différents.

Dans le cas d'un contrôle par file, l'algorithme d'arbitrage se réfère à une table d'éligibilité qui indique l'état de chaque file d'attente. Pour chaque cellule d'une file, cette table indique l'autorisation ou l'interdiction d'émettre cette cellule vers les ports de sortie du noeud. Il apparaît alors une évolution vers une synchronisation convergente.

Dans le cas d'un contrôle global, l'algorithme d'arbitrage se réfère à une table d'éligibilité qui indique l'état global des files. Cette table indique alors l'autorisation ou l'interdiction d'émettre une cellule pour l'ensemble des files d'attente. Une évolution tend ici vers une synchronisation aléatoire.

De ces problèmes de synchronisation, il résulte que de nombreuses cellules entrent en conflit lorsqu'elles doivent atteindre la même ressource, un port de sortie, au même moment. Elles congestionnent en fait

la ressource. Ce type de phénomène entraîne ainsi une diminution des performances de l'équipement.

Les figures 2a, 2b et 2c illustrent un problème de synchronisation dans le cas par exemple où le nombre d'entrée Ne du noeud est égal au nombre de sorties N_S.

La figure 2a illustre un cas sans problème, un processus d'arbitrage dans sa phase initiale par exemple. Dans la première interface, la file d'attente n° 1 est sélectionnée, dans l'interface n° X + 1, la file d'attente n° X + 1 est sélectionnée, enfin dans le dernier interface N_S., la file d'attente n° N_S. est sélectionnée, puis un nouveau cycle recommence pour toutes les priorités 1 des interfaces.

La figure 2b illustre le début d'une évolution vers un état de synchronisation. Dans le cas de la figure 2b, le port n° 1 de sortie est par exemple soumis à un blocage. Cela arrive notamment si le port de sortie est congestionné. Si deux cellules veulent au même instant sortir sur le même port, l'une d'elles pourra effectivement sortir et l'autre sera mise dans la file d'attente du port de sortie correspondant. Il n'y aura congestion que lorsque l'une des FIFO des ports de sortie aura atteint un seuil maximal de remplissage. Un contrôle de flux est ainsi activé et sera envoyé à tous les ports d'entrée pour que les algorithmes de choix intègrent cette information dans leur "table d'éligibilité". Le contrôle de flux ramené à l'entrée du noeud du réseau interdit alors à la première interface de servir la file d'attente n° 1, son algorithme d'arbitrage essaie alors de servir la file d'attente n° 2. A terme, il apparaît une congestion lorsqu'un seuil maximum est atteint par exemple entre les deux priorités 1 des deux premières interfaces qui tentent en même temps d'émettre une cellule vers le deuxième port de sortie du noeud 1 du réseau. Le port de sortie n° 2 interdisant son accès, les deux premières interfaces vont alors se synchroniser sur la file d'attente n° 3 comme l'illustre la figure 2c. A un instant, si plusieurs algorithmes sont synchronisés sur le n° 2 par exemple, la FIFO de sortie du port 2 va grandir et risque donc rapidement d'atteindre son seuil de congestion. Plus il y a d'algorithmes synchronisés, plus la probabilité de congestion augmente, d'où la tendance convergente vers la synchronisation. Un phénomène de cascade se poursuit jusqu'au moment où toutes les interfaces émettent une cellule sur le même port de sortie, le dernier port N_S par exemple. Il n'y a

pas de ressource physique comme un bus qui interdise à tous ports d'entrée d'émettre une cellule vers la même sortie. Ils émettent effectivement tous une cellule mais une seule d'entre elles sera émise en sortie, les autres seront mises dans la file d'attente du port de sortie visé.

De ce fait, désirant atteindre un même port de sortie au même moment, de nombreuses cellules créent donc une congestion. Ce type de phénomène entraîne ainsi une diminution des performances de l'équipement telle qu'exposée notamment précédemment.

Selon l'invention, afin d'éviter toute dégradation des performances, le procédé d'arbitrage rompt les cycles qui entraînent une synchronisation tout en garantissant l'équité entre les files d'attente. Pour cela le procédé introduit un aléa dans la sélection des files d'attente de même priorité. A chaque choix, selon le procédé selon l'invention, une file d'attente est par exemple tirée au sort parmi toutes les files d'attente éligible, sans se préoccuper par exemple des choix précédents. L'équité est garantie en moyenne. Ainsi, au lieu de choisir la file d'attente de rang X + 1, si celle-ci est éligible, après la file d'attente de rang X, les moyens d'arbitrage permettent de choisir parmi toutes les files d'attentes éligibles, quel que soit leur rang.

Une autre façon pour introduire un aléa selon l'invention consiste à tirer au sort parmi un nombre donné de files d'attente, par exemple à tirer au sort parmi N files d'attente éligibles suivant le rang de la dernière choisie. Ainsi si la précédente file d'attente choisie était X, et si le nombre total de files éligibles est N, alors les moyens d'arbitrage autorisent le tirage au sort d'un nombre R entre 0 et N de telle sorte que la Rième file d'attente éligible à partir de la Xième file soit sélectionnée. La file sélectionnée ne sera donc pas forcément R+X modulo Ns.

La figure 3 présente un mode de réalisation possible d'un dispositif pour la mise en oeuvre du procédé selon l'invention. Ce dispositif fournit le numéro de la file d'attente à servir. Il est associé à un circuit 201, 20P, 301, 30P de priorité donnée dans une interface d'entrée 2, 3. Il permet donc à l'interface de choisir une file d'attente parmi toutes celles d'une même priorité.

Selon la figure 3 un compteur 41 compte le nombre de files d'attente qui peuvent être élues. Pour cela le compteur 41 est relié à la table

d'éligibilité 42. Cette table est composée par exemple de N_S registres, N_S étant le nombre de ports de sortie du noeud du réseau. Le registre n° X indique par exemple l'état d'éligibilité de la file d'attente n° X par une information binaire, 1 signifiant par exemple éligible et 0 signifiant par exemple non éligible. Une file d'attente X est déclarée éligible si elle est non vide et si le port destinataire n'est pas congestionné. Le compteur 41 compte donc par exemple le nombre de bits égaux à 1 dans la table d'éligibilité 42. Le résultat est compris entre 0 et N_S. Un module 43 de génération de nombres aléatoires fournit des nombres aléatoires codés par exemple sur 16 bits, à partir par exemple d'un polynôme de degré 32. Les possibilités de choix sont dans ce cas très importantes. Le nombre aléatoire fourni est compris entre 0 et 1, 1 étant exclu. Il appartient donc à l'intervalle [0, 1[.

La sortie du compteur 41 indiquant le résultat du comptage de ce dernier et la sortie du générateur 43 de nombres aléatoires sont reliées aux deux entrées d'un multiplieur 44, ce dernier effectuant le produit de ces deux entrées, donc le produit du nombre de files d'attente éligibles par le nombre aléatoire fourni. Le nombre Ns de ports de sortie est par exemple tel qu'il peut être codé sur 5 bits. Le multiplieur 44 effectue un arrondi afin de présenter à sa sortie un nombre entier compris entre 1 et Ns. Le multiplieur 44 permet de réaliser un tirage aléatoire équitable d'un nombre entier compris entre 1 et N, avec N variable entre chaque tirage. Une autre solution aurait pu être de tirer au sort un nombre très grand compris entre 1 et 2^p-1 (tirage au hasard classique puisque les deux bornes sont fixes), puis exprimer le résultat en entier sur 5 bits modulo N (N = nombre de files éligibles) +1. On obtient ainsi un nombre compris entre 1 et N, mais la réalisation d'un modulo N (avec N non puissance de 2) n'est pas très aisé. De plus pour être certain d'avoir un tirage aléatoire équitable, it faut que p soit très grand. On voit donc les avantages en terme de simplicité et d'équité apportés par la solution proposée dans le dispositif de la figure 3.

La sortie du multiplieur 44 est reliée à une entrée d'un premier multiplexeur 45, l'autre entrée de ce multiplexeur ayant un état logique constant égal à 1. Ce multiplexeur 45 est commandé par un signal binaire M1. Sa sortie est reliée à une première entrée N d'un circuit de sélection 46.

La sortie d'un deuxième multiplexeur 47 est reliée à une deuxième entrée R du circuit de sélection 46. Une première entrée du deuxième

multiplexeur 47 est à l'état logique 0 alors que l'autre entrée est reliée à la sortie d'une mémoire 48 qui mémorise la sortie du circuit de sélection 46. Cette mémoire 48 mémorise en fait la dernière file d'attente servie.

Le circuit de sélection parcourt la table d'éligibilité à partir du rang R + 1, R étant le nombre affiché à sa deuxième entrée R. Il parcourt à partir de ce rang R + 1, N registres éligibles successifs, à l'état 1 par exemple, de la table d'éligibilité. Il présente à sa sortie le rang du dernier registre parcouru, par exemple R + 1 + N si tous les registres sont éligibles. Si le compteur 41 compte un résultat nul, il désactive le circuit de sélection 46 dont la sortie reste alors figée. Le compteur 41 est relié à une entrée d'activation du circuit de sélection 46. Le compteur indique si son résultat est nul par une information binaire. Cette dernière est aussi combinée avec la sortie du circuit de sélection 46 par l'intermédiaire d'une porte "et" 49. La sortie de cette porte est à zéro si le compteur est à zéro, dans ce cas aucune file d'attente n'est à servir. Le circuit de sélection 46 ayant été désactivé, le numéro de la dernière file d'attente servi est conservé en sortie du circuit. Il est donc possible de reboucler sur ce dernier quand des files d'attente seront de nouveau à servir. Si le compteur affiche un résultat non nul, la porte "et" 49 reproduit la sortie du circuit de sélection. La porte "et" 49 fournit en sortie le numéro de file d'attente à servir. Ce numéro est par exemple lu par un processeur qui active par exemple le registre ou la FIFO correspondante.

Les différents circuits, notamment le compteur 41 et le module 43 de génération de nombres aléatoires, sont par exemple synchronisés par un signal d'horloge non représenté, ou bien par un signal commandé par un processeur, chaque signal de synchronisation correspondant à une nouvelle sélection de file d'attente.

Les informations binaires M1, M2 qui commandent les deux multiplexeurs permettent de configurer le fonctionnement de l'ensemble.

Si M1M2 = 11, la première entrée N du circuit de sélection est forcée à 1 et la deuxième entrée R reproduit le dernier numéro de file d'attente servie. Après la file d'attente de rang R, celle de rang R+1 est ainsi sélectionnée, puis celle de rang R+2 et ainsi de suite. Un algorithme cyclique de type Round Robin est alors en fonctionnement. Aucun aléa n'est

introduit, c'est le cas décrit précédemment qui peut amener, dans certaines circonstances à une diminution globale des performances de l'équipement.

Si M1M2 = 00, la deuxième entrée R est forcée à zéro et la première entrée N du circuit de sélection 46 comporte le résultat du compteur 41. Après chaque sélection, une nouvelle file d'attente est alors choisie au hasard parmi toutes celles éligibles.

Si M1M2 = 01, la deuxième entrée R reproduit le numéro de la dernière file d'attente sélectionnée et la première entrée N du circuit de sélection 46 comporte le résultat du compteur 41.

Si M1M2 = 10, la première entrée N est fixée à 1 et la deuxième entrée R du circuit de sélection 46 est fixée à 0. La première file d'attente éligible à partir de la première file d'attente est alors sélectionnée.

Les circuits de la figure 3 pourraient par exemple être remplacés par un microcontrôleur intégrant les fonctions de tous ces circuits.

REVENDICATIONS

1. Procédé d'arbitrage entre des files d'attente (14) au niveau d'un noeud (1) d'un réseau de transmission numérique, caractérisé en ce que chaque file d'attente ayant un niveau de priorité attribué, il introduit un aléa (43) dans la sélection des files d'attente de même priorité.

2. Procédé selon la revendication 1, caractérisé en ce qu'une file d'attente (14) est tirée au sort parmi toutes les files d'attente autorisées à émettre une cellule.

3. Procédé selon la revendication 1, caractérisé en ce qu'une file d'attente (14) est tirée au sort parmi un nombre donné de files d'attente suivant le rang de la dernière file d'attente sélectionnée, et parmi les files d'attente autorisées à émettre.

4. Dispositif d'arbitrage entre des files d'attente (14) au niveau d'un noeud (1) d'un réseau de transmission numérique, caractérisé en ce qu'une file d'attente ayant un niveau de priorité attribué, il comporte des moyens de sélection (46) des files d'attente de même priorité et des moyens (41, 42, 43, 44, 45, 47, 48) pour introduire un aléa dans les moyens de sélection (46).

5. Dispositif selon la revendication 4, caractérisé en ce que les moyens pour introduire un aléa comportent un module (43) pour générer un nombre aléatoire, les moyens de sélection (46) opérant une sélection des files d'attente fonction de ce nombre aléatoire.

6. Dispositif selon l'une quelconque des revendications 4 ou 5, caractérisé en ce qu'il comprend des moyens (45, 47, M1, M2) pour configurer la sélection des files d'attente, avec ou sans aléa.

7. Dispositif selon l'une quelconque des revendications 4 à 6, caractérisé en ce qu'étant relié à une table d'éligibilité (42) des files d'attente, les moyens de sélection (46) sélectionnent une file d'attente parmi

toutes les files d'attente éligibles, une file d'attente éligible étant une file d'attente autorisée à émettre une cellule.

8. Dispositif selon l'une quelconque des revendications 4 à 6, caractérisé en ce qu'étant relié à une table d'éligibilité (42) des files d'attente, les moyens de sélection (46) sélectionnent une file d'attente parmi un nombre donné de files d'attente éligibles à partir de la dernière file d'attente sélectionnée, une file d'attente éligible étant une file d'attente autorisée à émettre une cellule.

9. Dispositif selon l'une quelconque des revendications précédentes, caractérisé en ce qu'il comporte au moins :

- un générateur (43) de nombre aléatoire compris entre 0 et 1
- un compteur (41) comptant les files d'attente éligibles dans une table d'éligibilité (42)
- un multiplieur (44) effectuant le produit du nombre aléatoire par le résultat du compteur (44)
- un circuit de sélection (46) parcourant un nombre N de registres éligibles successifs de la table d'éligibilité (42), le nombre N étant le produit du multiplieur (44), le circuit de sélection présentant en sortie le rang du dernier registre parcouru, chaque registre de la table d'éligibilité étant associé à une file d'attente, un registre éligible indiquant que sa file d'attente associée est autorisée à émettre une cellule, la sortie du circuit de sélection (46) indiquant le numéro de la file d'attente sélectionnée.

10. Dispositif selon la revendication 9, caractérisé en ce qu'il comporte un multiplexeur (45) intercalé entre le multiplieur (44) et le circuit de sélection (46), le multiplexeur étant commandé par un bit de configuration (M1) pouvant forcer la sortie du multiplexeur à 1.

11. Dispositif selon l'une quelconque des revendications 9 ou 10, caractérisé en ce qu'il comporte une mémoire (48) reliée en sortie du circuit de sélection (46), la sortie de la mémoire étant reliée à l'entrée d'un multiplexeur (47) dont la sortie est reliée à une deuxième entrée (R) du circuit de sélection (46), le multiplexeur (47) étant commandé par un bit de

configuration (M2) pouvant forcer sa sortie à 0, le circuit de sélection (46) parcourant la table d'éligibilité (42) à partir du rang correspondant au nombre affiché sur sa deuxième entrée (R).

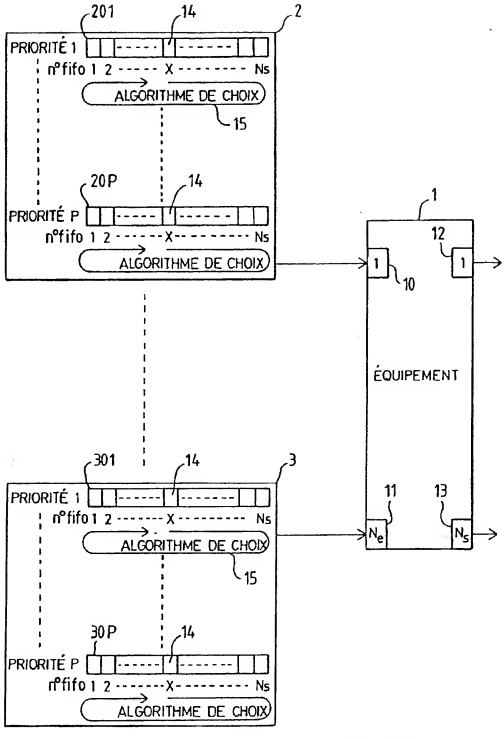

201
14
2
PRIORITÉ 1
n°fifo 1 2 ----- X --------- Ns
ALGORITHME DE CHOIX
15
20P
14
PRIORITÉ P
n°fifo 1 2 ----- X --------- Ns
ALGORITHME DE CHOIX
1
12
1
10
ÉQUIPEMENT
301
14
3
PRIORITÉ 1
n°fifo 1 2 ----- X --------- Ns
ALGORITHME DE CHOIX
15
11
13
Ne
Ns
30P
14
PRIORITÉ P
n°fifo 1 2 ----- X --------- Ns
ALGORITHME DE CHOIX

FIG.1

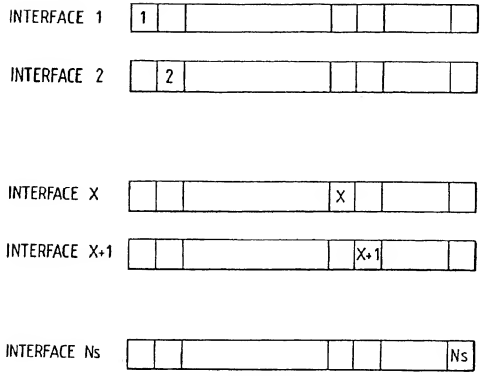

INTERFACE 1
1
INTERFACE 2
2
INTERFACE X
X
INTERFACE X+1
X+1
INTERFACE Ns
Ns

FIG.2a

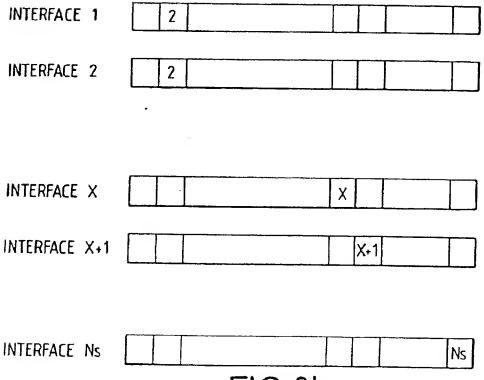

INTERFACE 1
2
INTERFACE 2
2
INTERFACE X
X
INTERFACE X+1
X+1
INTERFACE Ns
Ns

FIG.2b

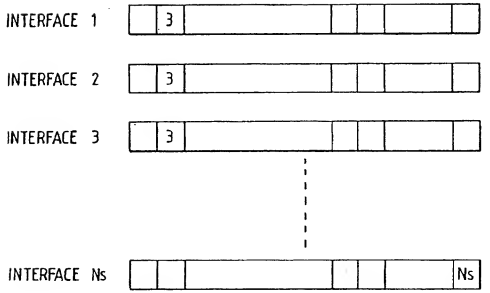

INTERFACE 1
3
INTERFACE 2
3
INTERFACE 3
3
INTERFACE Ns
Ns

FIG.2c

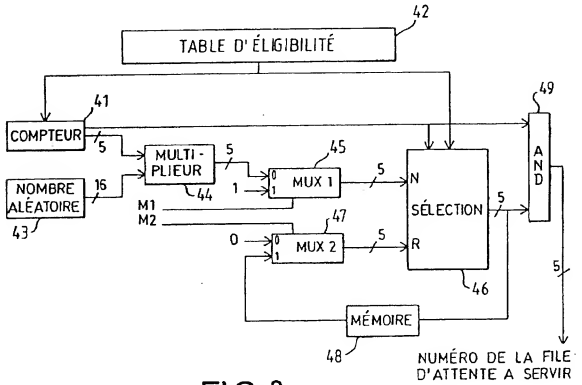

42
TABLE D'ÉLIGIBILITÉ
41
COMPTEUR
5
MULTI-
PLIEUR
5
44
45
MUX 1
5
N
SÉLECTION
5
R
46
49
A
N
D
5
NOMBRE
ALÉATOIRE
16
43
M1
M2
0
47
MUX 2
5
MÉMOIRE
48
NUMÉRO DE LA FILE
D'ATTENTE A SERVIR

FIG.3

REPUBLIQUE FRANÇAISE

INSTITUT NATIONAL
de la
PROPRIETE INDUSTRIELLE

RAPPORT DE RECHERCHE PRELIMINAIRE

établi sur la base des dernières revendications déposées avant le commencement de la recherche

2748620 ... national

FA 532352
FR 9605845

DOCUMENTS CONSIDERES COMME PERTINENTS		Revendications concernées de la demande examinée	
Catégorie	Citation du document avec indication, en cas de besoin, des parties pertinentes		
X	IEEE JOURNAL ON SELECTED AREAS IN TELECOMMUNICATION, vol. 6, no. 9, Décembre 1988, USA, pages 1587-1597, XP002023972 M.G. HLUCHYJ ET AL.: "Queueing in high-performance packet switching"	1,2,4,5, 7	
Y	* paragraphe II.A * * page 1589, colonne de droite, ligne 32 - ligne 41 * ---	6,9	
Y	SERVING HUMANITY THROUGH COMMUNICATIONS. SUPERCOMM/ICC, NEW ORLEANS, MAY 1 - 5, 1994, vol. VOL. 2, no. -, 1 Mai 1994, INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, pages 1184-1189, XP000438688 CHAO H J ET AL: "QUEUE MANAGEMENT WITH MULTIPLE DELAY AND LOSS PRIORITIES FOR ATM SWITCHES"	6	
A	* paragraphe 2.1 * ---	1-5,7-11	DOMAINES TECHNIQUES RECHERCHES (Int.Cl.6)
Y	"FRED FRAMEWORK III" 1988 , ASHTON-TATE CORPORATION XP002023973 * page 2-44 * --- -/--	9	H04L

1

Date d'achèvement de la recherche	Examinateur
28 Janvier 1997	Perez Perez, J

CATEGORIE DES DOCUMENTS CITES

X : particulièrement pertinent à lui seul
Y : particulièrement pertinent en combinaison avec un autre document de la même catégorie
A : pertinent à l'encontre d'au moins une revendication ou arrière-plan technologique général
O : divulgation non-écrite
P : document intercalaire

T : théorie ou principe à la base de l'invention
E : document de brevet bénéficiant d'une date antérieure à la date de dépôt et qui n'a été publié qu'à cette date de dépôt ou qu'à une date postérieure.
D : cité dans la demande
L : cité pour d'autres raisons

& : membre de la même famille, document correspondant

FORM 1503 03.82 (P0C13)

REPUBLIQUE FRANÇAISE

INSTITUT NATIONAL
de la
PROPRIETE INDUSTRIELLE

RAPPORT DE RECHERCHE PRELIMINAIRE

établi sur la base des dernières revendications déposées avant le commencement de la recherche

2748620
N° d'enregistrement national

FA 532352
FR 9605845

DOCUMENTS CONSIDERES COMME PERTINENTS

Catégorie	Citation du document avec indication, en cas de besoin, des parties pertinentes	Revendications concernées de la demande examinée
A	GLOBECOM'92. COMMUNICATION FOR GLOBAL USERS, INCLUDING A COMMUNICATIONS THEORY MINI CONFERENCE ORLANDO, DEC. 6 - 9, 1992, vol. 1 - 2 - 03, 6 Décembre 1992, INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, pages 104-110, XP000357769 DECINA M ET AL: "DATAGRAM SWITCHING NETWORKS FOR BROADBAND DATA COMMUNICATIONS" * paragraphe 2 * * figure 1 * -----	1,4

DOMAINES TECHNIQUES RECHERCHES (Int.Cl.6)

Date d'achèvement de la recherche: 28 Janvier 1997

Examinateur: Perez Perez, J

CATEGORIE DES DOCUMENTS CITES

X : particulièrement pertinent à lui seul
Y : particulièrement pertinent en combinaison avec un autre document de la même catégorie
A : pertinent à l'encontre d'au moins une revendication ou arrière-plan technologique général
O : divulgation non-écrite
P : document intercalaire

T : théorie ou principe à la base de l'invention
E : document de brevet bénéficiant d'une date antérieure à la date de dépôt et qui n'a été publié qu'à cette date de dépôt ou qu'à une date postérieure.
D : cité dans la demande
L : cité pour d'autres raisons

& : membre de la même famille, document correspondant

EPO FORM 1503 03.82 (P04C13)

WPIL

/ 6 *WPIL - ©Thomson Derwent - image*

AN - 2002-680077 [73]

TI - Device for controlling **queue** for communication between tasks and method thereof

DC - W01

PA - (HYNI-) HYNIX SEMICONDUCTOR INC

IN - KIM YS

NP - 1

NC - 1

PN - KR2002036050 A 20020516 DW2002-73 H04Q-001/30 1p *

AP: 2000KR-0065906 20001107

PR - 2000KR-0065906 20001107

IC - H04Q-001/30

AB - KR2002036050 A

NOVELTY - A device for controlling a **queue** for a communication between tasks and a method thereof are provided to enhance the performance of a system and manage a **queue** effectively by managing registrations of many **queues** being recognized in a main CPU board and an error message being asked in a general operating system in one place and transmitting a message to the final destination of each **queue** rapidly.

DETAILED DESCRIPTION - A task(100) performs a command inputted in an operator managing program. A processor(110) controls a program. A **queue** manager(120) supplies a message **queue** registration, a query function, and a transmission function. A **queue** thread(130) transmits/receives messages being transmitted and received to a remote place board to an IPC drive. A **queue table**(140) is used for storing and routing information of a **queue** being used in a system. A software block manufactures one's **queue** and requests a message **queue** registration to the **queue table**(140) through the **queue** manager(120). When the message is transmitted to other block, a query is processed for selecting a message **queue** for a transmission, and the **queue table**(140) is checked, and an ID of a destination **queue** is searched. A registered **queue** is designated as a block ID and an **interface** ID and used by attaching to a header in a message transmission. The **queue** thread(130) transmits a message transmission request to an IPC drive. The IPC drive transmits a message transmission request signal being transmitted from the **queue** thread(130) to the destination **queue**. In case that a responded block is in other unit, a message routing is performed. At this time, a mapping of the **queue table**(140) is performed by referring to a header of an IPC message. The destination **queue** reports a message result to the **queue** thread(130) again.(Dwg.1/10)

MC - EPI: W01-B09

UP - 2002-73

UP4 - 2002-11